Licentiate of Engineering Thesis

Measuring the Evolution of Automotive Software Models and Meta-Models to Support Faster Adoption of New Architectural Features

Darko Đurišić

University of Gothenburg

Division of Software Engineering
Department of Computer Science & Engineering
University of Gothenburg
Gothenburg, Sweden, 2015

Measuring the Evolution of Automotive Software Models and Meta-Models to Support Faster Adoption of New Architectural Features

Darko Đurišić

Technical Report No 127L
ISSN 1652-876X

Department of Computer Science & Engineering
Division of Software Engineering
University of Gothenburg
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2015.

To my loving Wife,
whose love and support is my constant source of inspiration,
and
to my Mom & Dad,
without who I would not be the person that I am today.

Abstract

Background: The ever-increasing amount of software in cars today combined with high market competition demands fast adoption of new software solutions in car development projects. One challenge in enabling such a fast adoption is to develop the architecture and models of the automotive software systems in a structured and controlled way.

Objective: The main objective of this thesis was to enable the fast utilization of new architectural features in automotive software models. This was achieved by developing methods and tools to analyze the evolution of the domain-specific meta-models that are used to define the language of software models and their features. In particular, we wanted to identify the underlying changes caused by meta-model evolution related to a specific set of architectural features and assess their impact on both the architectural models and modeling tools used by different roles (e.g., the Original Equipment Manufacturers, OEMs, and their suppliers) in the development process.

Method: We achieved our objective by conducting an action research project in close collaboration with the Volvo Car Group (Volvo Cars) and the consortium of the AUTOSAR standard, which aims to standardize the architecture of automotive software systems. This collaboration facilitates fast feedback from experts in the field on the problems, ideas and methods we developed in the course of this research, thereby enabling the validation of the research results and proposed methods in on-going development projects, i.e., their direct application in the industry.

Results: We identified the most suitable software measures for measuring the evolution of both the automotive software models and meta-models. The calculation and presentation of the measurement results were done with the support of two, newly-developed tools. We also developed a method for the automated identification of an optimal set of new architectural features that should be adopted in development projects to facilitate the decision-making process concerning the selection of which of these new features would be adopted.

Conclusion: We applied the developed methods and tools to the architectural models and meta-models used at Volvo Cars and concluded that they provide valuable input for the decision-making process concerning which new versions of the standardized meta-model should be used in different projects. We also concluded that these methods and tools can facilitate the assessment of the impact of adopting new architectural features on the different roles involved in the development process.

Acknowledgment

First of all I would like to express my deepest gratitude to my supervisor Miroslaw Staron. I was very fortunate to get a chance to work with Miroslaw since my Master study days. This served as an initial spark that lit my desire to pursue a PhD. Miroslaw's guidance and constant feedback before and during my PhD project was invaluable for my development as a researcher.

Next I would like to thank my co-supervisors Matthias Tichy and Jörgen Hansson whose ideas and comments significantly improved the quality of my research and included publications. I would also like to thank my managers at Volvo Stefan Andreasson and Hans Alminger for steering my development as an engineer and for supporting my wish to become an industrial PhD student.

I am very grateful to Urban Kristiansson, now retired Senior Technical Advisor from Volvo, for guiding me through the process of becoming a part of Volvo's Industrial PhD Program and ensuring my project funding. I am also very grateful to my colleagues Nicklas Fritzson, from the AUTOSAR team, and Ola Reiner, from the VSEM team, who kept this project alive with their ideas about the research and the application of the research results at Volvo.

Finally my inexpressible appreciation goes to my family who is the most important thing in my life. First and foremost I want to thank my wife Bojana for unlimited encouragement in my moments of doubt and for sharing my moments of success. I hope that I will find a way to make it up to you for all the work you had to carry out yourself during my traveling and writing. I am also very grateful to my parents Slobodan and Mirjana whose support in my personal and professional life decisions was an important source of motivation.

The research presented in this thesis was conducted within the QuaSAR@car research project which is funded by Volvo Cars, as part of the Volvo Industrial PhD Program (VIPP), and by Swedish Governmental Agency for Innovation Systems (VINNOVA), under the grant no. 2013-02630.

List of Publications

Included papers

This licentiate thesis is based on the following papers:

[A] D. Durisic, M. Nilsson, M. Staron and J. Hansson, "Measuring the Impact of Changes to the Complexity and Coupling Properties of Automotive Software Systems", *Journal of Systems and Software, vol 86, no 5, pp 275-1293, 2013*

[B] D. Durisic, M. Staron, M. Tichy and J. Hansson, "Evolution of Long-Term Industrial Meta-Models - A Case Study of AUTOSAR", *Proceedings of the 40th Euromicro Conference on Software Engineering and Advanced Applications, pp. 141-148, 2014*

[C] D. Durisic, M. Staron, M. Tichy and J. Hansson, "Quantifying Long-Term Evolution of Industrial Meta-Models - A Case Study", *Proceedings of the International Conference on Software Process and Product Measurement, pp. 104-113, 2014*

[D] D. Durisic, M. Staron, M. Tichy and J. Hansson, "Identifying Optimal Sets of Standardized Architectural Features - A Method and its Automotive Application", *Proceedings of the 11 International ACM SIGSOFT Conference on Quality of Software Architectures, 2015*

[E] D. Durisic, M. Staron, M. Tichy and J. Hansson, "ARCA - Automated Analysis of AUTOSAR Meta-Model Changes", *Proceedings of the 7th International Workshop on Modeling in Software Engineering, 2015*

Other publications

The following paper is published but not appended to this thesis due to the content overlap with the first paper.

[a] D. Durisic, M. Staron and M. Nilsson, "Measuring the Size of Changes in Automotive Software Systems and their Impact on Product Quality", *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement, pp 10-13, 2011*

Contents

Chapter 1

Introduction

The use of software in cars dates from the end of the 1970s and the start of the 1980s when microcomputer applications began to be used for engine control [1]. Today, software plays an important role in almost every function of a modern car. Automotive functionalities, such as adaptive cruise control, the lane departure warning system and automated parking support, depend primarily on the use of software. The analysis of Charette [2] suggests that it takes more than 100 million lines of code to put a premium car on the road today, which is more code than some passenger airplanes require to operate their avionics and on-board support systems. The trend of increasing amounts of software in modern cars is expected to continue [3] due to new, future functionalities, including autonomous drive and car-to-car communication.

In order to stay competitive, car manufacturers (Original Equipment Manufacturers, OEMs) must constantly incorporate new automotive functionalities in their car models. This requires fast software innovation cycles in the car development projects, which in turn require changes in the software used in the actual cars (the automotive software). The automotive software is at a high abstraction level described by the software architectural model; the latter usually must be updated to support the addition of new car functionalities and enable changes in the automotive software. These updates represent the evolution of the automotive software architecture and its model.

The automotive architectural model is comprised of a set of predefined architectural features that represent its building blocks. These features are described by domain-specific meta-models, the availability of which determines the technology that can be used in the cars themselves, for example, the use of Ethernet as a communication medium between distributed parts of the automotive software system. As the automotive software technology evolves, so do the domain-specific meta-models in order to support the new architectural features, which are then used to model different architectural components of the system. This means that the models and the domain-specific meta-models evolve in parallel with each other.

This thesis addresses the problem of cost-efficient management of the domain specific meta-model evolution in car development projects. Our main objective is to facilitate the decision-making process concerning which of the new architectural features will be used in future cars in order to support the modeling of new functionalities. We achieved this objective by developing tools and methods to automate the analysis of meta-model changes caused by specific architectural features.

The proposed methods are based on a number of software measures that we defined to evaluate the evolution of the architectural models and meta-models. These measures can quantify the changes between different meta-model versions for a set of new architectural features. The measurement results can also be used to visualize the evolution of both the architectural models and meta-models with respect to a number of properties, such as size, complexity and coupling. We used these results to assess the impact of different architectural features on the modeling tools used by various roles (e.g., the OEMs and their suppliers) involved in the automotive software development process.

We applied the proposed methods and tools to the evolution of the AUTomotive Open System Architecture (AUTOSAR) meta-model [4] and the architectural models developed at Volvo Cars. The AUTOSAR meta-model is a standardized meta-model used to define the language for the architectural models of the automotive domain and is fundamental to the development of automotive modeling tools. We showed that the methods can be used to facilitate the decisions regarding which new releases of the AUTOSAR standard or subsets of its new features will be adopted in on-going or future car development projects.

This chapter is structured as follows: Section 1.1 describes the role of models and meta-models in software development; Section 1.2 describes the automotive modeling environment and the role of AUTOSAR in this environment; Section 1.3 describes both the importance and use of software measurement in this thesis; Section 1.4 defines our research questions and describes the contribution and impact on industry of each paper included in this thesis; Section 1.5 describes the methodology we employed to analyze the data and obtain the results; finally, Section 1.6 summarizes our future research plans.

Chapters 2-6 present the individual papers included in this thesis. Each paper is independent and represents one study that addressed one or several, smaller research questions.

1.1 Modeling and meta-modeling

Modeling plays an important role in the development of large software systems because it can reduce their complexity by raising the abstraction level. This abstraction level is achieved by specifying what the system does rather than how it does this [5]. Models and meta-models are the two most important concepts involved in any modeling environment. Based on the definitions of Bézivian et al. [6], a model represents a simplified representation of a software system that has been created for a specific purpose, whilst a meta-model represents the model specification using a specific modeling language.

The word "meta" indicates that something was done twice, in this case the modeling; for example, the meta-model represents "a model of the model" and the meta-meta-model represents "a model of the meta-model". Applying this logic several times creates a meta-modeling hierarchy, which is also referred to as the meta-pyramid [7] and is generally accepted to consist of four layers:

1. The **M3 layer**: a meta-meta-model that defines the modeling concepts
2. The **M2 layer**: a meta-model that defines the language specifications
3. The **M1 layer**: a model that defines the instance specifications
4. The **M0 layer**: an object that defines the system instances

These layers are connected by the instantiation mechanism, i.e., each layer represents an instance of the layer above; for example, *M0* is an instance of *M1*, except for the top layer *M3*, which is considered to be an instance of itself. Generally, one element is an instance of another element if the instantiating element defines the characteristics of that instance element, and the instance element defines the specific details of these characteristics. For example, an instantiating element may specify that a *Signal* has a data-type and an instance of this *Signal* may define an integer data-type.

A model is an instance of the meta-model, whilst a meta-model is an instance of the meta-meta-model. According to the strict meta-modeling principle [8], all model elements on one layer of a meta-modeling hierarchy are instances of the model elements of the layer above except for the top layer. No other relationships that cross the boundaries of different layers are possible. The *instanceOf* relationship, however, is not transitive; for example, the *M1* model element will only receive characteristics from the *M2* meta-model elements and not from the *M3* meta-meta-model elements [9].

The meta-modeling hierarchy described above is also accepted by the Object Management Group (OMG) [10] which is considered a *de facto* standard for meta-modeling (see Figure 1.1).

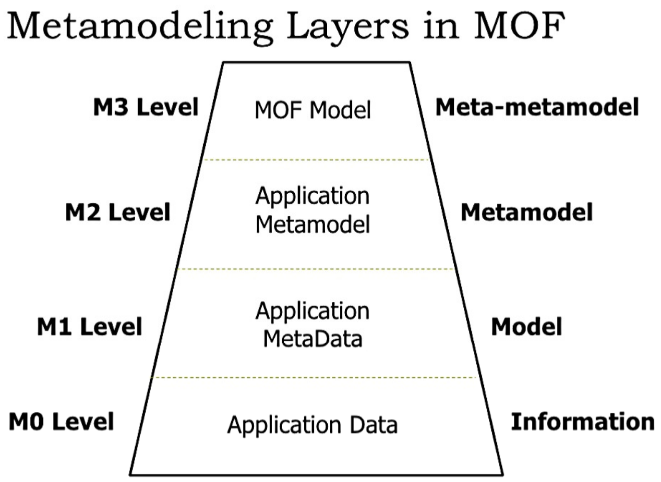

Figure 1.1: The Meta-Object Facility (MOF) layers

The Meta-Object Facility (MOF) [11] resides at the top of the hierarchy. This is a meta-meta-model that defines the general modeling concepts used by meta-models on the *M2* layer. A frequently used meta-model on the *M2* layer is UML (Unified Modeling Language). The actual UML models reside on the *M1* layer and their actual execution at run-time resides on the *M0* layer.

The *M3* and the *M2* layers represent language specifications. Four important concepts of one language specification can be distinguished, namely the abstract syntax, concrete syntax, static semantics (their well-formedness) and dynamic semantics (also refereed to as semantics). The abstract syntax describes the structural essence of the language, for example, a general concept used for all purposes. The concrete syntax renders the abstract concepts for a specific purpose (e.g., using graphical notation) and there can be more than one concrete syntax for an abstract syntax. The static semantics impose a set of constraints on the abstract syntax in order to apply the abstract concepts. Finally, the dynamic semantics give meaning to the syntax notation of the language. A meta-modeling environment must at least specify the abstract syntax, but can also specify the concrete syntax and the static and dynamic semantics [12].

Two different types of meta-models, namely the linguistic and ontological meta-models, can be distinguished based on the two forms of the *instanceOf* relationship described earlier [5]. The primary purpose of the linguistic meta-models is to define a language (the abstract syntax) for the specification of models instantiating this meta-model. The primary purpose of the ontological meta-models is to define the meaning of the models. The ontological meta-models can be used as domain-specific language definitions instantiating a linguistic meta-model to provide both the syntax and static semantics for one modeling environment (please refer to the description of the domain-specific meta-models below).

These two forms of the *instanceOf* relationship are also responsible for the meta-modeling property known as dual classification [13], where the instances on the *M0* layer are both ontological instances of the *M1* classes (which in turn are instances of the *M2 Class*) and linguistic instances of the *M2 Object*. Therefore, the illustration of the meta-modeling layer hierarchy shown in Figure 1.1 breaks the strict schema principle as the *M0* instances directly instantiate (linguistically) the *M2 Object*, i.e., the *M1* layer is omitted.

One way of solving this problem is to consider the strict meta-modelling principle for only one of the two forms of *instanceOf* relationships, i.e. either ontological (from the system modelers perspective) or linguistic (from the tool implementers and language designers perspectives). In terms of the linguistic approach, ontological modeling can, to some extent, be achieved using stereotypes [14, 15]. Another approach would be to represent the linguistic and the ontological instantiations using two dimensions, where the ontological instantiation is depicted horizontally and the linguistic instantiation is depicted vertically; an example is shown in Figure 1.2.

The *O0* and *O1* vertical layers represent the ontological layers, while the *L0*, *L1* and *L2* horizontal layers represent the linguistic layers. We can see that the *User Object* on the linguistic layer *L1* and the ontological layer *O0* is both a linguistic instance of the generic *Object* on the *L2* and an ontological instance of the concrete *User Class* on the *O1* layer. Moreover, the actual *Runtime* object on the linguistic layer *L0* only occurs on the ontological layer *O0*; thus, we can only form a mental picture of it on the *O1* layer.

Finally, not all modelers interpret the relationship between the models and the meta-models in this way, as demonstrated by Kühne [16], because it depends on the exact interpretation of the word ”meta”. So far, we have referred

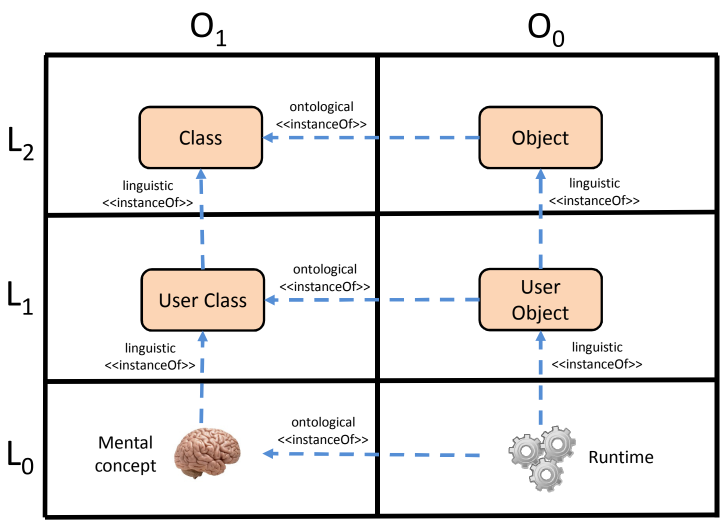
O1
O0
L2
L1
L0
Class
Object
ontological <<instanceOf>>
linguistic <<instanceOf>>
linguistic <<instanceOf>>
User Class
User Object
ontological <<instanceOf>>
linguistic <<instanceOf>>
linguistic <<instanceOf>>
Mental concept
ontological <<instanceOf>>
Runtime

Figure 1.2: A two-dimensional representation of the layers [5, 13]

to a meta-model as "a model of the model using the *instanceOf* relationship"; for example, the executing code is an instance of the UML class diagram. We can, however, also define a model of the model using the *representedBy* relationship, where both models reside on the same meta-modeling layer; for example, a UML class diagram is a model of the source code. Figure 1.3 shows an example of the *instanceOf* and the *representedBy* relationships.

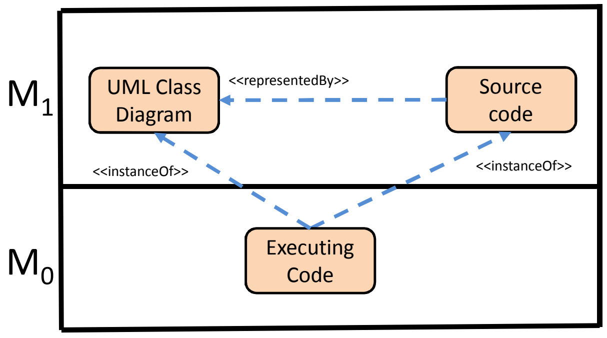

Figure 1.3: The *representedBy* and *instanceOf* relationships

Generally it is agreed that the relationship between the models and the meta-models is based on the *instanceOf* relationship, whilst the *representedBy* relationship indicates the model-to-model transformation.

1.1.1 Domain-specific meta-models

Meta-modelling plays an important role in the development of the description languages suitable for modeling of specific domains [17], including automotive, telecommunications and avionics. A domain-specific model represents an abstract representation of the system of a particular domain, whilst a domain-specific meta-model defines the syntax and the semantics of the domain-specific models instantiating this meta-model [18]. Several domain-specific meta-models may focus on specifying different subject areas [19], such as data, work-flows and tasks.

The syntax of one domain-specific meta-model includes both the abstract and concrete syntaxes, which define the language constructs, and the entities and their relationships within the abstract syntax, respectively. The semantics, however, include only static semantics (e.g., each signal needs to have a data-type), which can be achieved by specifying different constraints, such as the range of the relationships and OCL (Object Constraint Language) constraints [20], to ensure their well-formedness. On the other hand, dynamic semantics (e.g., all signals must meet their timing requirements) cannot be specified in the domain-specific meta-model; usually they are achieved by providing supporting specifications in a natural language. In the case of UML-based, domain-specific meta-models, UML profiles with the defined stereotypes and tag definitions can also be a powerful mechanism of customizing the concrete syntax and static semantics of the UML meta-model for a specific domain.

Any one, domain-specific modeling environment may contain an arbitrary number of layers; in practice, there can be more than the four layers described in Figure 1.1. These layers must evolve to support the addition of new system functionalities, and the evolution of one layer may require the evolution of some or all of the other layers [21]. For example, in order to express new modeling solutions on the *M1* layer, the *M2* layer must evolve in order to describe how to model these new solutions. Similarly, evolution of the *M2* layer may require evolution of existing models of the *M1* layer in order to maintain conformity between the *M1* and *M2* layers.

The challenge of parallel evolution of both the models and meta-models of one domain-specific modeling environment, therefore, must be addressed by system modelers and meta-modelers, although this combined model-meta-model evolution can be automated to a certain extent [22], for example, the automated re-factoring of existing *M1* models in order to conform to a new *M2* meta-model [23].

1.1.2 Domain-specific modeling tools

System modelers of one domain-specific modeling environment usually rely on software modeling tools to create and update the models and generate code based on these models. Since the development of large software systems often involves many roles potentially using different modeling tools in the development process, a smooth exchange of models between these roles can be somewhat challenging. Nevertheless, a smooth exchange can be enabled by defining and gaining consensus from all roles for a domain-specific meta-model, which is then fundamental to the development of all tools used in a specific modeling environment. This is based on the assumption that if two modeling tools adopt the same model structure defined by the meta-model, they can exchange a number of software models that comply with this meta-model [24].

Since the modeling tools are based on a commonly-accepted meta-model, its evolution may significantly impact all the tools in a specific modeling environment. Such tools tend to be developed based on their own meta-models (e.g., to support graphical representations) so evolution of the domain-specific meta-models direct impacts the importers and exporters of the compliant models, as well as the meta-models used by these tools. The parallel evolution of both the domain-specific and tool-specific meta-models, therefore, is another problem that must be addressed by tool suppliers in the development of large software systems.

1.1.3 Architectural models and features

Software architecture represents a set of design decisions made about a system. A model that captures some or all of these design decisions can be regarded as the architectural model [25]. An architectural model defines a number of architectural components responsible for the execution of different system functionalities.

Based on these definitions, the possibility of utilizing the architectural components and their interactions in a specific way to achieve certain semantics can be considered to be an architectural feature, for example, the possibility of transmitting system signals between different parts of the distributed system via an Ethernet cable. The architectural models and their features are expressed using a general modeling language, which has a dual classification in both the linguistic and ontological layers. For example, in order to model the transmission of concrete signals via a concrete Ethernet cable, a domain-specific meta-model must define the "signal" and "Ethernet" components and their connection, i.e. how the signals are transmitted via the Ethernet.

1.2 Automotive software development

The development of automotive electrical systems (software and hardware) relies on the use of different models [26] and is usually based on the V-model process shown in Figure 1.4.

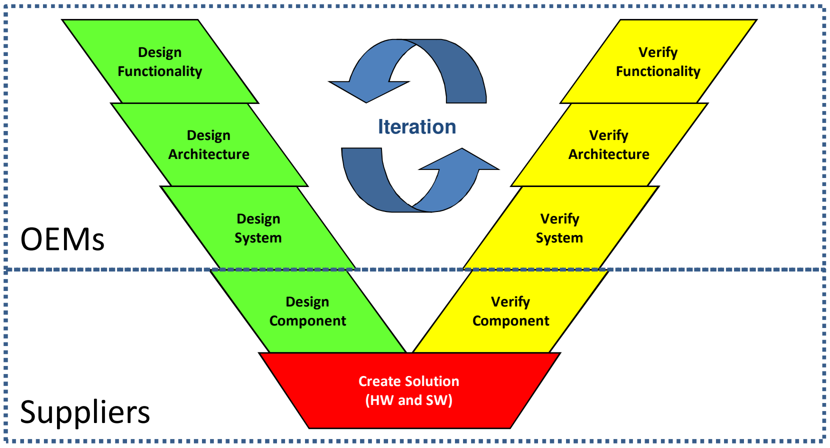

Figure 1.4: The development of automotive electrical systems

The left side of the V-model determines the design of the electrical system, whilst the right side is responsible for system verification. The first step in the development process is to define a model for a number of vehicle functionalities, such as an early collision warning. The system architects then define a general, architectural model of the system by creating a topology of the Electronic Control Units (ECUs) connected via electronic buses and allocating the vehicle functionalities onto these ECUs. The architectural models are refined in the system design step, where a number of software (and hardware) components controlling the execution of vehicle functionalities are modeled and allocated onto different ECUs. These steps and their verification matches on the right side of the "V" are usually done by the OEMs. In this thesis, we focus on the architectural and system models, i.e., on their specific design steps.

The design, implementation and verification of both the software and hardware for one ECU are usually done by the suppliers who are organized into different tiers according to their responsibilities, such as implementation of the software components and the middleware software and hardware delivery. The development process based on the aforementioned V-model is usually iterative and consists of several iterations where new vehicle functionalities are added and the architecture and system designs are updated accordingly.

The AUTOSAR standard was introduced in 2003 as a joint partnership of the OEMs and their suppliers in order to facilitate the development of automotive electrical systems based on the V-model. Today, AUTOSAR consists of more than 150 global partners [4] and, therefore, is considered to be a *de facto* standard for automotive software system development. Thus, it is hardly feasible these days for OEMs to develop automotive electrical systems that are not based on the AUTOSAR standard because fewer suppliers are available, which significantly increases development costs.

The AUTOSAR standard defines the following main objectives:

1. Increased re-use of architectural components developed by the automotive software suppliers in different car projects (within one or multiple OEMs). This allows cheaper software components to be used with higher quality (as these components are tested in several car projects).

2. Standardization of the middleware and hardware layers, which enables software designers and implementers to focus more on the design and implementation of complex vehicle functionalities.

3. Standardization of the exchange format for the architectural models, which enables a smooth model exchange between a number of software modeling tools developed by different tool suppliers.

These AUTOSAR objectives are achieved using three-layer ECU architecture. The top layer (*Application software*) defines the software components and their exchange points and is closely associated with the architecture system design steps of the V-model. The middle layer (*Run-time environment*) controls communication between different software components abstracting the fact that they may be allocated to different ECUs. If they are allocated to different ECUs, transmission of the respective signals on the electronic buses is done by the bottom layer (*Basic software*), which is generally responsible for "basic" ECU functionalities, such as communication with the hardware, buses, memory and diagnostic services.

Both the *Run-time environment* and *Basic Software* layers are completely standardized by AUTOSAR, such that AUTOSAR provides detailed specifications for all architectural components of these layers. This standardization, together with the clear distinction between the *Application software*, *Run-time environment* and *Basic software* layers, allow the system designers and implementers to specifically focus on the development of vehicle functionalities without being distracted by middleware and hardware design. The application software and basic software architectural components are developed often by different suppliers who specialize in either one of these areas, referred to as Tier 1 and Tier 2 suppliers, respectively.

In order to standardize the exchange format for the architectural models of the *Application software* layer between the OEMs and the Tier 1 and 2 suppliers, AUTOSAR defines a domain-specific meta-model that specifies the language for these models. Based on the AUTOSAR meta-model, the modeling tools of OEMs and suppliers can exchange the architectural models efficiently, thereby minimizing any tool interoperability issues.

1.2.1 AUTOSAR modeling environment

The AUTOSAR modeling environment consists of five layers, the names of which are taken from the AUTOSAR *Generic Structure* specification [27]:

1. The **ARM4**: MOF 2.0, e.g., the MOF Class
2. The **ARM3**: UML and AUTOSAR UML profile, e.g., the UML Class
3. The **ARM2**: Meta-model, e.g., the SoftwareComponent
4. The **ARM1**: Models, e.g., the WindShieldWiper
5. The **ARM0**: Objects, e.g., the WindShieldWiper in the ECU memory

The AUTOSAR modeling environment defines five meta-layers in an attempt to incorporate both the linguistic and ontological *instanceOf* relationships that are of importance to the tool suppliers and the system designers, respectively. This means that subsequently the strict meta-modeling principle is not achieved. To solve this problem, we characterized the AUTOSAR meta-modeling hierarchy using the two-dimensional representation shown in Figure 1.5 and represent the linguistic instantiation (corresponding to the MOF layers) vertically and the ontological layers horizontally.

The *ARM2* and *ARM1* layers represent the ontological layers *O1* and *O0*, respectively, and both reside on the same linguistic layer *L1* (MOF layer *M1*). The *ARM4* and *ARM3* layers represent the linguistic layers *L3* and *L2* (MOF layers *M3* and *M2*), respectively. The *ARM0* layer corresponds to the linguistic layer *L0* (MOF layer *M0*) and resides on the ontological layer *O0* because it represents the execution of the concrete object in an ECU. This representation conforms to the strict meta-modeling principle as the *instanceOf* relationships never cross the boundaries of more than two layers.

The *ARM4* and *ARM3* layers define the abstract syntax, while the *ARM2* layer (referred to as the AUTOSAR meta-model) defines the concrete UML

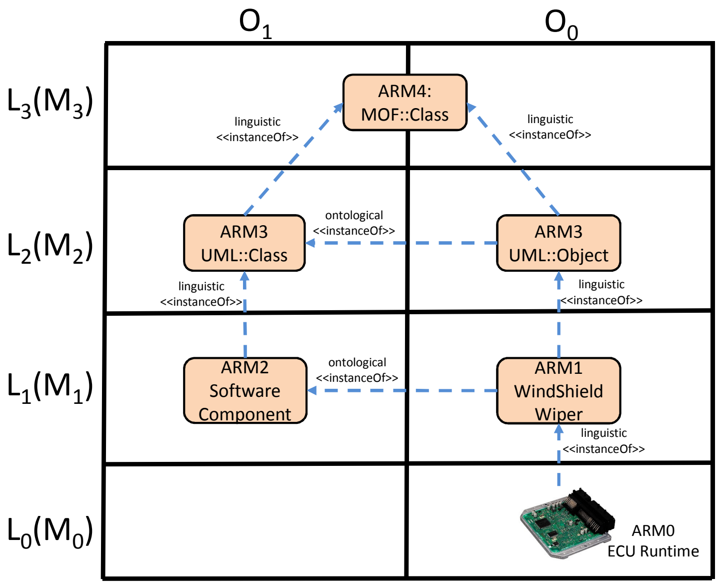
O1
O0
L3(M3)
L2(M2)
L1(M1)
L0(M0)
ARM4: MOF::Class
linguistic <<instanceOf>>
linguistic <<instanceOf>>
ARM3 UML::Class
ontological <<instanceOf>>
ARM3 UML::Object
linguistic <<instanceOf>>
linguistic <<instanceOf>>
ARM2 Software Component
ontological <<instanceOf>>
ARM1 WindShield Wiper
linguistic <<instanceOf>>
ARM0 ECU Runtime

Figure 1.5: The AUTOSAR meta-model layers

syntax and the static semantics of the AUTOSAR models residing on the *M1* layer. The AUTOSAR meta-model also uses the AUTOSAR UML profile from the *ARM3* layer, which specifies the used stereotypes and tags. All other constraints and dynamic semantics are defined in the natural language specifications that support the AUTOSAR meta-model.

1.2.2 AUTOSAR meta-model

The AUTOSAR meta-model (*ARM2* layer) is divided into a number of top-level packages, referred to as "templates". Each template is standardized and defines how to model one specific part of the automotive electrical system. For example, the *SoftwareComponentTemplate* defines how to model software components and their communication, whilst the *SystemTemplate* defines how to model ECUs (i.e., the allocation of software components onto the ECUs, ECU connections using electronic buses, etc.) and their communication (i.e., the signals exchanged on the electronic buses).

Based on the AUTOSAR templates residing on the *ARM2* layer, the OEMs and their suppliers can create proprietary models on the *ARM1* layer instantiating these templates for different parts of the automotive system. This is true for all models except for those instantiating the *ECUCDefinitionTemplate*. The latter models contain a definition of the ECU *Basic software* configuration parameters and are standardized by AUTOSAR. In order words, AUTOSAR provides the standardized *ARM1* models for the configuration parameters of all *Basic software* modules. The actual values of these parameters are defined in the models instantiating the *ECUCDescriptionTemplate* and they reference their corresponding definitions (see the example shown in Figure 1.6).

On the smallest granularity, standardized models of the *ECUCDefinition-Template* are divided into a number of packages, where each package contains the configuration parameters of one *Basic software* module. On the highest granularity, these models are divided into different logical packages, including ECU communication, diagnostics, memory access and IO access. Moreover, in papers B, D and E of this thesis, we have analyzed the evolution of the standardized models of the *ECUCDefinitionTemplate*, together with the evolution of the meta-model templates, and focused on the logical packages of ECU communication (the role of *ECU communication configurators*) and diagnostics (the role of *Diagnostic configurators*).

The values of certain configuration parameters from the *ECUCDescriptionTemplate* models can be automatically derived from models of other templates, such as the *SoftwareComponentTemplate* or *SystemTemplate*. This process is called ”upstream mapping” and is referred to as this in the thesis papers. The role of *Upstream mapping tool developers* defined in Paper B, for example, is to implement tool support for the automatic derivation of parameter values of the *ECUCDescriptionTemplate* models from the *SoftwareComponentTemplate* and *SystemTemplate* models.

A simplified example of different AUTOSAR templates (*ARM2*) and their models (*ARM1*) is shown in Figure 1.6. The gray color represents the elements standardized by AUTOSAR, whilst the light blue color represents proprietary elements modeled by different OEMs.

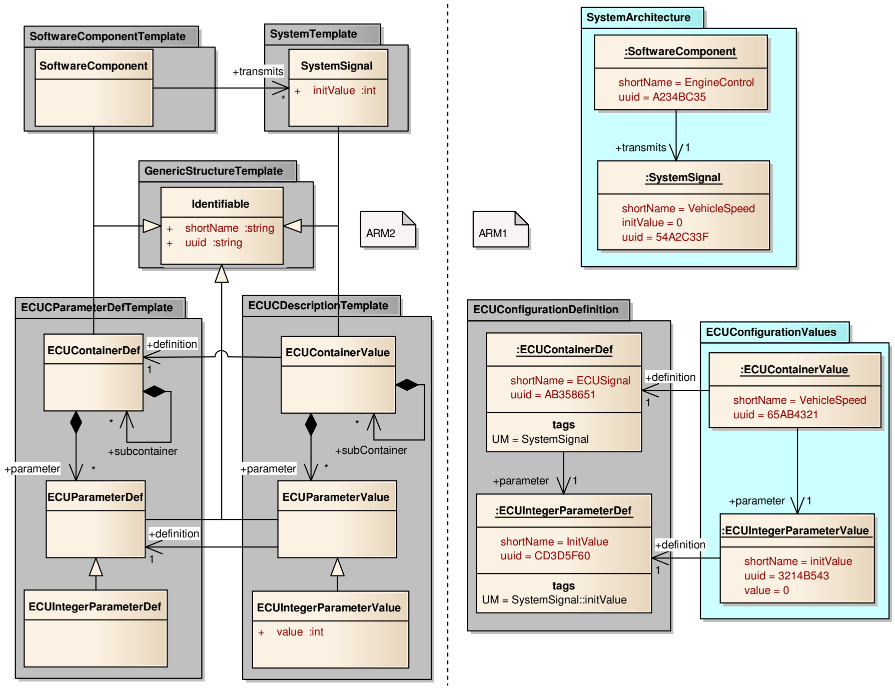

Figure 1.6: An example of the AUTOSAR templates and their models

On the *ARM2* layer, we can see the elements of the five templates. The *ECUCDefinitionTemplate* specifies the modeling of the definition of ECU parameters and containers of these parameters with an example of the integer parameter. The *ECUCDescriptionTemplate* specifies the modeling of the container and parameter values. The *SoftwareComponentTemplate* and the *SystemTemplate* specify a simplified modeling of the signal transmission by the software components. Finally, all elements of the templates shown are inherited from a common element in the *GenericStructureTemplate* (*Identifiable*), which provides both a name and unique identifier (UUID) for these elements.

The standardized model of the *ECUConfigurationDescription* can be seen on the *ARM1* layer, which shows the *ECUSignal* container with the *InitValue* integer parameter. These two elements both have a tagged value with the name *UM*, denoting Upstream Mapping. The *UM* tagged value for the *ECUSignal* container refers to the *SystemSignal* from the *SystemTemplate.* The *UM* tagged value for the *InitValue* parameter refers to the *initValue* attribute of the *SystemSignal.* This implies that for every *SystemSignal* instance in the *SystemArchitecture* model, one container instance in the *ECUConfigurationValues* will be created with an integer parameter instance. The value of this parameter instance will be equal to the *initValue* attribute of that *SystemSignal* instance (0 in our example). This derivation of the ECU configuration parameter values from the system architectural models can be automated using modeling tool support.

1.2.3 AUTOSAR models

Managing the evolution of the AUTOSAR meta-model is essential in enabling the incorporation of new features in automotive architectural models and thus in car development projects. This is because these models must be fully compliant with the AUTOSAR meta-model to ensure a smooth model exchange between a number of roles in the development process. Thus, they are also referred to as the AUTOSAR models. In order to update the AUTOSAR models with new architectural features, all roles (e.g. the OEMs and their suppliers) must update their modeling tools according to new versions of the AUTOSAR meta-model.

The AUTOSAR models are structural models and they should not be confused with behavioral models that are used to describe concrete ECU functionalities, for example, auto-braking when a pedestrian is detected on the vehicle's trajectory. These behavioral models are usually developed in Matlab Simulink tool from which the actual source code (C code) can be generated. Certain parts of the ECU source code related to the functionality of the middleware layer, *Run-time environment* and the configuration of the *Basic Software* modules (referred to as the aforementioned "upstream mapping" process) can also be generated from the AUTOSAR models.

The AUTOSAR models are expressed using Extensible Markup Language (XML) and are validated by the AUTOSAR XML schema generated from the AUTOSAR meta-model. This process is referred to as the XML Meta-data Interchange (XMI) and is a standardized process for data exchange between different roles in the development process created by OMG [28].

1.3 Software measurement

Measurement in software engineering plays an essential role not only to assessing a variety of software system quality attributes, such as reliability, maintainability and efficiency [29], but also to estimating the implementation cost and effort to support different features. According to the measurement theory [30], measurement is a process in which numbers (or symbols) from the mathematical world are assigned to different entities from the empirical world to describe the entities according to defined rules. A measure represents a variable to which a value is assigned as a result of the measurement [31]. In the papers of this thesis, we also refer to measures as metrics [32].

The measurement theory describes how to construct measures (metrics) and formalize their mapping from the empirical to the mathematical world, i.e., empirical relations between entities are mapped to mathematical relations so that they can be analyzed. For example, if two software systems (A and B) are related by the empirical relation "more complex than", we can define a measure of their complexity (c), where measurement results are related by the mathematical relation ">". The mapping between the empirical and mathematical relations, therefore, is defined as:

$$c(A) > c(B) => A \textit{ more complex than } B \tag{1.1}$$

Measurement results can be represented on different scales and different relations between the results are possible depending on the scale [30]:

- **Nominal scale** - only a relation of equivalence possible (A = B)
- **Ordinal scale** - a nominal relation + greater/smaller than (A > B)
- **Interval scale** - an ordinal relation + difference computation (A − B)
- **Ratio scale** - an interval relation + ratio computation (A / B)
- **Absolute scale** - like ratio, but counts the number of items

In order to structure the software measures according to their objectives, the Goal Question Metric (GQM) approach proposed by Basili et al. can be used [33]. The GQM defines the measurement as a mechanism that helps to answer a variety of questions about the software process and products. It defines the measurement model on three levels: Conceptual (goals), Operational (questions) and Quantitative level (metrics) as shown in Figure 1.7:

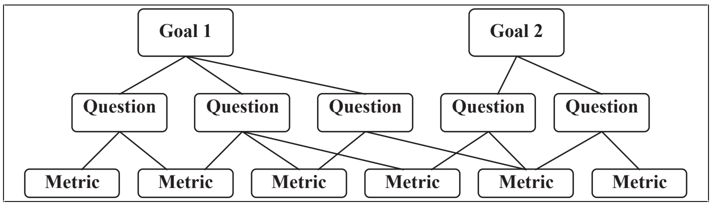

Figure 1.7: The Goal Question Metric (GQM) levels [33]

The goal is defined for one object (e.g., a product or process) with respect to its quality attributes for a specific purpose (e.g., an evaluation) and from a specific perspective (e.g., that of a system designer). A set of questions for one goal is used to specify how the goal will be assessed by characterizing the object to be measured. Finally, the measures represent the quantitative data associated with every question that enable the question to be answered.

1.3.1 Measurement process

The *ISO/IEC 15939* standard for the measurement process in software engineering [34] defines the process as a set of activities that are required to specify: (i) what information is needed for the measurement, (ii) how the measures are made and measurement results are analyzed, and (iii) how the results are validated. Additionally, the measurement process specifies how to build the measurement products, although this area is beyond the scope of this thesis. A simplified measurement process is shown in Figure 1.8:

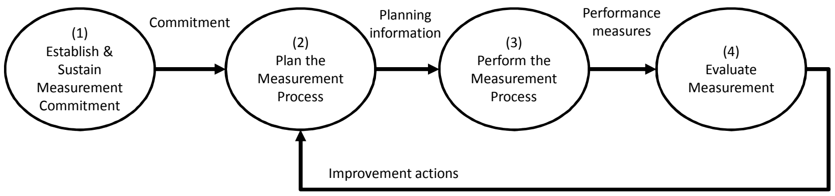
(1) Establish & Sustain Measurement Commitment
Commitment
(2) Plan the Measurement Process
Planning information
(3) Perform the Measurement Process
Performance measures
(4) Evaluate Measurement
Improvement actions

Figure 1.8: Measurement process activities [34]

Activity (1) defines the scope of the measurement and who will execute it. Activity (2) elaborates on the measurement plan, such as what is to be measured (i.e., which entities and their quality attributes), what information is needed (i.e., the reason for the measurement), which measures will be used and on what scale, and the criteria for evaluating the measurement results. Activity (3) describes how the data will be collected and analyzed. Finally, activity (4) describes how the measures and the measurement process will be evaluated based on the defined criteria of activity (3). This process is defined as iterative in order to improve both the measures and measurement process based on the results of the evaluation in activity (4).

One important segment of the planning activity (3) is to ensure that the measures are clearly defined in order to avoid different interpretations of how the measurement has been done (see the measurement errors of different implementations of the commonly known lines of code measure [35]). The measures will be defined based on the conceptual model, which is used to describe the entities in the empirical world [36] to ensure that the metrics can satisfy the required information need.

Software measures are usually defined using either set theory or algebra expressions. In order to prevent a definition of a measure using one of these two approaches from becoming too complex, however, alternative approaches can be taken, such as using pseudo-code snippets. For example, the complexity (c) of one software component (x) can be defined using algebra as follows:

$$c(x) = \sum_{i=1}^{n} r_i(x) * \sum_{i=1}^{n} t_i(x);$$

$$r_i(x) = \begin{cases} 1, & \text{if x receives } sig_i \\ 0, & \text{otherwise} \end{cases} ; t_i(x) = \begin{cases} 1, & \text{if x transmits } sig_i \\ 0, & \text{otherwise} \end{cases}$$

where n represents the total number of signals and sig_i the signal with serial-number i. Using set theory, this same result can be achieved by defining two sets $Sin(x)$ and $Sout(x)$:

- $Sin(x) = \{sin_1(x), sin_2(x), ..., sin_\alpha(x)\}$ - a set of signals received by software component x.
- $Sout(x) = \{sout_1(x), sout_2(x), ..., sout_\beta(x)\}$ - a set signals transmitted by software component x.

The complexity would then be calculated as follows:

$$C(x) = |Sin(x)| * |Sout(x)|$$

Finally, the corresponding pseudo-code could look like this:

```
int Complexity(Component x)
{
    int Sin = 0;
    int Sout = 0;

    foreach (Singal in ReceivedSingals(x))
        Sin = Sin + 1;

    foreach (Singal in TransmittedSingals(x))
        Sout = Sout + 1;

    return Sin * Sout;
}
```

An important part of the measurement process is to validate the defined software measures. Two different types of validation can be performed [37]: a theoretical validation to answer "*Are we measuring the right attribute?*" and an empirical validation to answer "*Is the measure useful?*".

The theoretical validation ensures that a measure does not violate the properties of the measured entity [38]. This can be achieved by assessing whether the measure satisfies certain theoretical criteria. For example, there must be at least two entities for which the measure yields a different result, measuring the same entity twice yields the same results, measuring two entities can yield the same result, etc. Additionally, Briand et al. [39] classify the measures according to five attributes (size, length, complexity, coupling and cohesion) and define a set of properties required to measure each attribute. These properties can be used to group measures according to different properties and validate that they do indeed measure an intended attribute.

The empirical validation ensures that the measurement results are consistent with expected values [38]. This can be achieved by discussing the results with experts who work with the measured entity to ensure that they are consistent with their expectations (e.g., code complexity should decrease after re-factoring it). Statistical analysis techniques can also be used to validate the relationship between two attributes if the measurement goal was to explain one attribute which, for example, cannot be measured by measuring another attribute. An example of this is the use of a correlation analysis based on historical data to validate the link between code complexity and the number of faults.

1.3.2 The use of software measurement in this thesis

The results presented in this thesis rely heavily on the use of software measures and measurement results. The automotive architectural models and meta-models represent the scope of the measurements, and our main information need was to assess their evolution with respect to a number of properties, including changes, size, complexity and coupling. Since available, off-the-shelf measures do not use the specific modeling characteristics of the automotive domain, we defined our own set of measures for this purpose.

The measures are defined following the GQM approach based on the appropriate conceptual models. Since we performed several studies to address different goals and research questions, we followed the GQM approach because it helped us to structure the measures in order to address all of the goals and questions. For example, in Paper B, we defined a measure of meta-model change in order to analyze the evolution of domain-specific meta-models, whilst in Paper C, we defined a measure of meta-model size, length, complexity, coupling and cohesion in order to assess the impact of the meta-model evolution on different roles in the development process.

Clearly defining both the goals and questions for each study also enabled us to re-use the logic behind certain measures used for measuring different entities in order to address similar questions and goals about these entities. For example, the goal of our complexity and coupling measures defined in Paper A was to monitor the evolution of architectural models and was based on the research question: *What is the trend in the complexity and coupling evolution of these models?*. One of the goals of Paper C was to monitor the evolution of the same properties for the domain-specific meta-models based on the research question: *What is the impact of meta-model evolution on different roles in the development process?*.

Consequently, we re-used the logic behind our complexity and coupling measures for software components of the architectural models (the definition of which was based on the conceptual model presented in Paper A) and redefined the measures so that they would apply to meta-classes of the domain-specific meta-models (based on the conceptual model presented in Paper C). In particular, the measures defined in Paper A were based on the interaction between different software components using signals, whilst those of Paper C were based on the interaction between different meta-classes using associations.

Some of our measurement results are presented on the ratio scale (e.g., model and meta-model complexity) whilst others are shown on the absolute scale (e.g., the number of meta-model changes). The majority of measures are defined using either set theory or algebra. We defined the measure of the complexity and coupling of the architectural models in Paper A using algebra, and the measure of size, length, complexity, coupling and cohesion of the meta-models in Paper C using set theory. Data collection was automatic because we had developed tools to measure the properties of both the models and meta-models based on the appropriate conceptual model.

The measures used in our studies are validated both theoretically and empirically. The theoretical validation was based on the properties defined by Briand et al. [39] for the size, length, complexity, coupling or cohesion measures depending on the classification of the measures. The empirical validation was based on applying the measures on project data from Volvo Cars and the AUTOSAR consortium.

The complexity and coupling measures for monitoring the evolution of the architectural models were applied to a number of software components and ECUs from two evolving electrical systems at Volvo Cars. The measurement results were presented to experts who were able to confirm the results based on their knowledge about actual system changes. The measure of meta-model change was applied to a number of AUTOSAR meta-model releases, and the results of the measurements matched the change documentation of AUTOSAR. Finally, the measures for monitoring the size, length, complexity, coupling and cohesion of domain-specific meta-models were applied to a number of historical releases of the AUTOSAR meta-model and the choice of the most suitable measures was based on statistical analysis (correlation and principal component analysis).

A summary of the measures used in our studies is shown in Table 1.1.

Table 1.1: The most important measures used in the studies of this thesis

Measure name	Measure goal	Defined	Used
Complexity metric	Monitoring the complexity evolution of the automotive software systems	Paper A	Paper A
Package coupling metric (SW components)	Monitoring the coupling evolution of the automotive software systems	Paper A	Paper A
Number of changes (including Number of changed elements and attributes)	Estimating the effort / cost of adopting a new meta-model version / feature and finding the optimal set of features to be adopted	Paper B (textual description)	Papers B, D, E
Number of classes and Number of attributes	Monitoring the size of meta-model evolution	Paper C	Papers B, C, E
Average depth of inheritance	Monitoring the length of meta-model evolution	Paper C	Papers B, C, E
Fan-in, Fan-out and Fan-in-out	Monitoring the complexity of meta-model evolution	Paper C	Papers C, E
Package coupling metric (classes) and Coupling between objects	Monitoring the coupling of meta-model evolution	Paper C	Paper C, E
Package cohesion metric and Cohesion ration	Monitoring the cohesion of meta-model evolution	Paper C	Papers C, E

1.4 Research questions and contributions

The main objective of this thesis was to enable faster adoption of new architectural features in automotive software development projects. Therefore, we defined the following, key research question:

Q *How can the evolution of architectural models and meta-models related to a set of new architectural features be managed efficiently?*

In order to answer our key research question, we divided it into several, smaller research questions that we addressed in different studies. As a motivational study, we analyzed the evolution of architectural models of automotive software systems in order to understand the complexity impact of adding new car functionalities to the systems. Moreover, our objective for this first step was to define a set of software measures capable of quantifying the complexity and coupling properties of the architectural models. Thus, we defined the following research questions:

Q1 *How can a change in the complexity and coupling of automotive architectural models be measured?*

Q2 *What is the trend in the complexity evolution of architectural models during the lifetime of one automotive software system?*

The results showed that the changes in the models were significant and a deeper understanding of the source of these changes was required. Our next step, therefore, was to analyze the changes between different versions of the automotive domain-specific meta-model, which specifies how different architectural features are modeled. This analysis was facilitated by defining the following research questions:

Q3 *How can a measure of change between different domain-specific meta-model versions be defined?*

Q4 *Can this measure of change be used to monitor the evolution of domain-specific meta-models?*

The results showed that different meta-model changes usually impact the models and tools developed by different roles in the automotive software development process, so the meta-model changes for each role were analyzed separately. For this reason, we also defined the following research questions:

Q5 *What major roles exist in the automotive software development process?*

Q6 *Which parts of the domain-specific meta-model are relevant for the modeling tools used by these roles?*

Q7 *Which roles are affected most by the evolution of the domain-specific meta-model?*

The results showed that monitoring only the number of changes may not suffice for an accurate impact assessment of the new, domain-specific meta-model versions and their features on different roles. This is because some changes only affect certain roles, whilst others affect multiple roles. The latter changes are considered to be more severe, such that monitoring other meta-model properties, for example, complexity, coupling and cohesion, is also important. This led us to define the following research questions:

Q8 *Which measures are the most suitable to measure the evolution of domain-specific meta-models with respect to their size, length, complexity, coupling and cohesion properties?*

Q9 *How can the results of different software measures be combined in order to assess the impact of meta-model changes on different roles?*

Up until this point, the analysis showed that not all architectural features from the new domain-specific meta-model versions are relevant for all roles involved in the development process. This means that a subset of features is relevant to a subset of roles, such that the roles usually must decide which new architectural features to adopt in the development projects. Therefore, as it is also important to analyze the changes in new meta-model versions related to a subset of their features, we defined the following research questions:

Q10 *How can the evolution of domain-specific meta-models with respect to their different architectural features be quantified?*

Q11 *How can an optimal set of new architectural features that are to be adopted in the software development projects be selected efficiently?*

We combined the answers to research questions *Q1-Q11* in order to answer our main research question *Q*. Specifically, we used (i) the identified roles, (ii) the defined software measures, and (iii) the method for selecting an optimal set of architectural features that are to be adopted in the development project in order to perform the role-based analysis of domain-specific meta-model evolution related to its architectural features.

1.4.1 Paper contributions

All research questions (*Q1-Q11*) are addressed in one of the papers of this thesis. Table 1.2 shows which questions were addressed in which paper.

Table 1.2: Published contributions to this thesis

Paper	Addressed research question
Paper A	Q1, Q2
Paper B	Q3, Q4, Q5, Q6, Q7
Paper C	Q8, Q9,
Paper D	Q10, Q11

In Paper A, we concluded that the architecture of automotive software systems is very complex; this complexity is constantly increasing with the addition of new car functionalities. Therefore, measuring the increase in complexity and coupling of these architectures is an important aspect in reducing faults and increasing system longevity. The measures proposed in this paper can be used for this purpose. To automate both the measurement process and presentation of measurement results, we developed a tool (*QTool*), which can be used during the evolution of automotive software systems in order to indicate when and where a set of architectural changes can be performed to reduce system complexity.

In Paper B, we concluded that a role-based analysis of changes between different domain-specific meta-model versions is valuable in estimating the effort needed to adopt certain meta-model versions in development projects. We defined a method that can successfully monitor the evolution of domain-specific meta-models (i.e., the changes) for different roles in the development process. We also identified the major roles in the automotive development process that are most affected by changes in the domain-specific meta-model used in the automotive industry.

In Paper C, we assessed a number of software measures that are applicable for measuring the evolution of domain-specific meta-models with respect to five properties: size, length, complexity, coupling and cohesion. We also showed how to combine the results of these measures in order to assess the impact of adopting new version of the domain-specific meta-models on the modelling tools used by different roles in the development process.

In Paper D, we defined a method (*MeFiA* - Meta-model Feature Impact Assessment) that can identify an optimal set of new architectural features that are to be adopted in the development projects. The method is based on the impact of these features on domain-specific meta-models used in the development and feature prioritization. In particular, we identify which meta-model changes are related to which architectural feature.

In addition to papers A-D, which answer the research questions *Q1-Q11*, Paper E has also been included in this thesis. Paper E presents a tool (ARCA) that can automatically perform the analysis described in papers B, C and D, thereby answering research questions *Q3-Q11*. The *ARCA tool* can be used to facilitate the evolution of automotive software architectures related to the adoption of new architectural features, as defined in the main research question *Q*. In particular, the tool can assess the impact of meta-model changes on modeling tools used by different roles. This impact is used in the analysis of which new meta-model versions or a subset of their features will be adopted in the development process of different projects.

1.4.2 Industrial contributions

All results presented in this thesis are directly deployed at Volvo Cars by means of incorporating the two implemented tools *QTool* and *ARCA* in the development process of automotive software systems.

The *QTool* implements the complexity and coupling measures presented in Paper A and is primarily used by the automotive Software Architecture Testers. The tool is used during the evolution of an automotive software system after the addition of new functionalities in order to analyze their impact on the complexity and coupling properties of different architectural components (e.g., sub-systems, ECUs and domains). If the results are unsatisfactory, the architectural components must be re-designed so as to reduce coupling and increase the ECU cohesion. This could be done by re-allocating a subset of the software components (i.e., functionalities) onto other ECUs, which might immediately result in a reduced number of signals on the electronic buses.

The *ARCA* tool described in Paper E implements both the measures and methods that are defined in papers B-D, and is used by Automotive System Designers. There are three, main situations in which this tool can be used:

1. To analyze changes between AUTOSAR meta-model releases

The analysis is done by four different teams at Volvo Cars for four main roles in the AUTOSAR-based software development process. The System Architects (i) and System Designers (ii) analyze the impact of the changes on the *Application Software Designers* role. The Signal Database Team (iii) analyzes the impact of the changes on the *ECU Communication Designers* role. Finally, the AUTOSAR Team (iv) analyzes the impact of the changes on the *ECU Communication Configurators* and *ECU Diagnostic Configurators* roles, as well as on other parts of the ECU configuration, such as non-volatile memory and encryption library.

The aim of this analysis is to facilitate the decisions concerning which new AUTOSAR release will be adopted in the development process as part of a cost-benefit analysis (Volvo Cars currently uses AUTOSAR release *4.0.3* and is analyzing the impact of switching to one of the *4.1.1-4.2.1* AUTOSAR releases). The *ARCA* tool can be used to estimate the effort required to implement the changes for each analyzed role and will be complemented by an analysis of the actual need for adopting new AUTOSAR releases.

2. To analyze changes related to specific AUTOSAR features

The analysis is done primarily by the System Architects at Volvo Cars with support from the System Designers and AUTOSAR Team. It includes the impact assessment of adopting each new AUTOSAR feature on the four main roles in the AUTOSAR based software development (*Application Software Designers*, *ECU Communication Designers*, *ECU Diagnostic Configurators* and *ECU Communication Configurators*) and identification of the optimal set of features to be adopted using the *MeFiA* method.

The aim of this analysis is to facilitate the decisions concerning which new AUTOSAR features will be supported as part of a cost-benefit analysis (Volvo Cars currently analyzes the impact of adopting new features from the AUTOSAR release *4.2.1*). The *ARCA* tool can be used to estimate the effort needed to implement the changes for the analyzed feature and will be complemented by an analysis of the actual need for adopting this feature.

3. To analyze changes between SVN versions of the AUTOSAR meta-model during the development of one release

This analysis is done by the AUTOSAR Team at Volvo Cars with the aim of continuously following-up changes in the AUTOSAR meta-model in order to have sufficient time to influence their standardization (the tool is currently used to analyze changes for the AUTOSAR release *4.2.2*).

For example, if an accepted change in the AUTOSAR meta-model removes one meta-element that is widely used at Volvo Cars, the change will be discussed again in the AUTOSAR consortium involving representatives from Volvo Cars before its official release. This is particularly important for companies that are not AUTOSAR core partners (companies driving the AUTOSAR standard) because they tend not to be represented in all AUTOSAR groups, making it more difficult for them to be aware of all of the changes. Additionally, core partners have a certain period of time (i.e., a few weeks) before the official release to review and block any critical changes. This is not, however, the case with other AUTOSAR partners, who must invest additional effort to influence the changes during the implementation phase of one release. In such a situation, the *ARCA* tool could be a valuable aid.

As well as the implementation of these thesis results at Volvo Cars, the *ARCA* tool functionality related to a comparison of different SVN versions of the AUTOSAR meta-model for a particular new feature is planned to be adopted in the daily change management process of AUTOSAR. The idea is to present the actual changes in the AUTOSAR meta-model in corresponding *Bugzilla*[1] implementation tasks so that the reviewer can confirm that all changes are really intended. This information will also be used to generate the change documentation between different AUTOSAR releases and traceability of the AUTOSAR meta-model changes to the *Bugzilla* implementation tasks.

1.5 Research methodology

Research methodology describes the systematic process of data collection and analysis in order to achieve the desired conclusions. The general research methodology used in the studies of this thesis is action research. Using only one research method, however, does not usually suffice for the combined work of experts and researchers [40]. Therefore, we reported each cycle of the action research as one case study.

In this section, we summarize the theory of the action research and the case study, demonstrate how we described each action research cycle as one case study and finally, discuss the validity of our results.

1.5.1 Theory of action research

The term "action research" was first introduced by Kurt Lewin in 1946 [41] as a research approach in social sciences intended to improve inter-group relations. Lewin acknowledged the necessity of the existing scientific approaches of acquiring general knowledge (e.g., by conducting surveys) and situation specific knowledge (e.g., by conducting experiments), but questioned their value if they had no practical use ("*Research that produces nothing but books is not sufficient.*"). Therefore, he called for a research type that immediately leads to social actions in order to help experts deal with the actual problems. One definition of action research commonly used was designed by Rapoport [42]:

"Action research aims to contribute both to the practical concerns of people in an immediate problematic situation and to the goals of social science by joint collaboration within a mutually acceptable ethical framework."

Therefore, the main aim of action research is to improve the practice and increase collaboration between the experts and scientists [43]. This is further elaborated by Susman et al. [44] who define a circle of five steps around an evolving client system (e.g., a company facing practical problems) that will be conducted during the action research process, as shown in Figure 1.9.

[1] A tool used by AUTOSAR for issuing and documenting change requests.

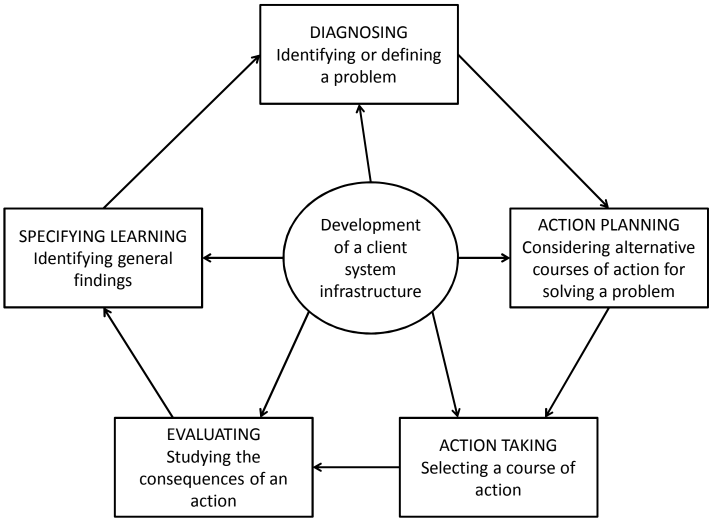
DIAGNOSING
Identifying or defining a problem
ACTION PLANNING
Considering alternative courses of action for solving a problem
ACTION TAKING
Selecting a course of action
EVALUATING
Studying the consequences of an action
SPECIFYING LEARNING
Identifying general findings
Development of a client system infrastructure

Figure 1.9: The cyclical process of action research [44]

The diagnosing phase focuses on identifying the practical work problems faced by the experts from companies involved in the action research project. The next phase involves the initial planning of actions that must be taken in order to solve the identified problems. These actions are usually undertaken by the corporate experts, although action research can also be involved. The evaluation phase assesses the result of applying the actions at the companies by analyzing if the identified problems still exist. If not all defined problems are solved, the entire process is repeated taking into consideration inputs from the previous phases in re-defining the actions to be taken. The last phase of the cycle is driven by the action researcher and is concerned with identifying overall findings that are important for the generalization of the results to create scientific knowledge applicable to other, similar cases (e.g., at other companies facing similar problems).

The action researcher does not have to be involved in all five phases. Depending on the involvement of the action researcher in different phases, we can distinguish between the following main types of action research [45]:

- **Diagnostic** - action researcher is involved in diagnosing the problem
- **Empirical** - action researcher is involved in evaluating the results
- **Participant** - action researcher is involved in diagnosing and planning
- **Experimental** - action researcher is involved in nearly all phases

1.5.2 Theory of case study

A case study is classified as an empirical research method [46] and focuses on the examination of a real-world situation making it quite suitable for industrial evaluations. Yin [47] defines a case study as an iterative process consisting of five phases, as shown in Figure Figure 1.10:

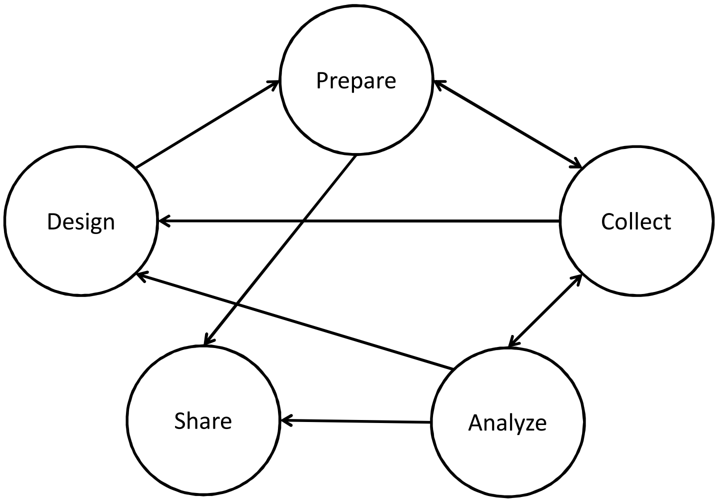

Figure 1.10: The five phases of a case study [47]

Here, we focus on the design, collect and analyze phases. The design phase consists of the following five components [47]: (i) Research questions, (ii) Study propositions, (iii) Unit of analysis, (iv) Linking data to prepositions and (v) Criteria for interpreting the findings.

The design of each study begins with a clear definition of the research questions. Providing answers to these questions is the main objective of a case study. In order to understand how to achieve this goal, however, the scope of the study is defined together with the identification of elements that are to be examined (the study propositions). Next, the unit of analysis (i.e., the "case"), the elements of which are to be examined, is defined. The data obtained from this examination are then linked to the propositions in order to answer the defined research questions using, for example, statistical analysis. Finally, the criteria for data interpretation are defined in order to indicate when the obtained results can be considered valid, for example, by defining the statistical significance if statistical methods are used to analyze data.

The data collection phase can include both qualitative and quantitative data collection methods [46]. Qualitative data can be obtained by analyzing documentation, performing observations, conducting interviews, etc. Interviews are especially common in analyzing industrial cases because they provide quick answers to question from experts. They can be formal with a precisely-defined set of questions, informal relying on a casual discussion with experts [48] or semi-formal where questions are pre-defined, but can be deviated from during the interview [49]. Quantitative research represents the analysis of numerical data in order to explain a certain phenomenon [50]; quantitative data are usually obtained by measurement.

Data can also be analyzed using different methods, such as pattern matching (comparing the empirical pattern with one or several predicted patterns) and explanation building (e.g., using theoretically proven concepts) [47]. Quantitative data can be analyzed using a number of statistical methods, including correlation analysis and time-series analysis.

1.5.3 One action research cycle as a Case study

Each action research cycle can be seen as one case study that aims to answer a set of research questions related to the study context relevant for this cycle. Understanding this context is crucial for defining the actions that can solve the identified problems. Additionally, the case study analysis provides an immediate possibility to evaluate the results of the actions on the chosen unit of analysis. On the other hand, action research methodology provides the possibility to collate the results of different case studies into one method (i.e., a set of actions) that can address the general problem. Action research also emphasizes the need to discuss how to generalize the proposed method, thereby raising the level of its scientific significance.

The definition of research questions is part of the diagnosing phase of the action research. The choice of the unit of analysis, the study propositions and the description of how to obtain the data and link them to the chosen propositions (e.g., a method definition) are all part of the action planning phase. The action-taking phase includes the application of the proposed method on the unit of analysis and data collection. Finally, data evaluation corresponds to the data analysis phase of the case study with focus on the validation of the performed actions (e.g., the proposed method). The fact that each new case study may define new research questions and use different units of analysis is important for advancing the knowledge between different action research cycles because the results of one cycle can be used as input for the next cycle.

In the studies included in this thesis, we conducted an experimental action research project at Volvo Cars where the author of the thesis was the action researcher working at the company. The general aim of the project was to develop methods and tools to address the main research question *Q*, which was divided into smaller research questions *Q1-Q11* as explained in Section 1.4. We conducted four action research cycles, each based on one case study with the aim of addressing one or several smaller research questions. Each case study was documented in one of the thesis papers. The first case study served as a motivational study and the results of the second case study served as input to the subsequent case studies.

Case study 1 (research questions *Q1* and *Q2*)

The action researcher was part of the Software Architecture Testing Team at Volvo Cars. The diagnosing phase was performed at the start when the problem of how to monitor the complexity of the automotive architectural models was defined. In the planning phase, it was decided that the architectural models used at Volvo Cars would be the unit of analysis. The outcome of the case study was a method (based on two measures) and a tool for monitoring the complexity of the automotive architectural models. The method was validated based on its application on two evolving architectures and the tool was included in the software architecture testing process at Volvo Cars.

Case study 2 (research questions *Q3*, *Q4*, *Q5*, *Q6* and *Q7*)

The action researcher was part of the AUTOSAR Team at Volvo Cars. The diagnosing phase was performed at the start when the problem of how to monitor the evolution of domain-specific meta-models for different roles involved in the development was defined. In the planning phase, it was decided that the AUTOSAR meta-model would be the unit of analysis. The main outcome of this case study was a method (based on a measure of change) for monitoring the evolution of domain-specific meta-models for different roles. The method was validated based on its application on a number of AUTOSAR meta-model releases for seven main roles in the automotive software development process. The roles were identified using semi-structured interviews with a number of engineers from automotive OEM and software supplier companies.

Case study 3 (research questions *Q8* and *Q9*)

The action researcher was part of the AUTOSAR Team at Volvo Cars. The diagnosing phase was performed at the start when the problem of which software measures would be used to assess the impact of changes in domain-specific meta-models on different roles was defined. In the planning phase, it was decided that the AUTOSAR meta-model would be the unit of analysis. The identified roles and defined data model in the Case Study 1 served as an input to this analysis. The main outcome of Case Study 3 was a method (based on a combination of measures) for assessing the impact of changes in domain-specific meta-models on different roles. The method was validated based on its application to historical releases of the AUTOSAR meta-model. Correlation and principal component analyses were used to identify the most suitable measures.

Case study 4 (research questions *Q10* and *Q11*)

The action researcher was part of the AUTOSAR Team at Volvo Cars. The diagnosing phase was performed at the start where the problem of how to identify an optimal set of new architectural features to be adopted in the development projects was defined. In the planning phase, it was decided that the AUTOSAR features would be the unit of analysis. The identified roles and defined measure of change in Case Study 1 served as an input to this analysis. The main outcome of this case study was a method (based on the measure of change) for identifying an optimal set of architectural features to be adopted in development projects. The method was validated based on its application on 14 new features of the AUTOSAR release *4.2.1*.

Based on the methods defined in case studies 2, 3 and 4, we developed a tool presented in Paper E. The tool was included in the process of analyzing the impact of new AUTOSAR meta-model releases and their features on different roles involved in software development at Volvo Cars.

Since AUTOSAR and the AUTOSAR meta-model were used as a unit of analysis in case studies 2, 3 and 4, the action researcher was appointed to be one of the representatives of Volvo Cars in the AUTOSAR consortium. The aim was to ensure direct contact between the action researcher and experts from the AUTSOAR consortium. This in turn enabled fast feedback loops in the action planning and result evaluation phases by a number of experts in the field, including OEMs and different types of suppliers.

1.5.4 Research validity

We followed the principles of Baskerville et al. [51] to increase rigor during the action research project, with a particular focus on:

- Maintaining collaboration between the action researcher and experts during all phases of the action research.
- Promoting iterations of different phases, particularly action planning, execution and evaluation of the results.
- Ensuring generalization of results in the specifying learning phase.

According to Cook and Campbell [52], four types of validity threats to empirical studies conducted in the area of software engineering must be considered. We explain how we addressed each of these in our four case studies (four action research cycles) below.

Internal validity

Internal validity is concerned with the results of the analysis not being casual, i.e., the relationship between the measured properties and the outcome should not be random. The most severe threat to the internal validity in our studies was related to the measurement process which was performed by developing two software tools for calculating the measures. In order to ensure internal validity, we performed detailed testing of the tools using smaller examples before employing them for the main measurements.

External validity

External validity concerns generalization of results and is one of the most prominent threats to action research validity (i.e., the applicability of the results to other companies facing similar problems) [51]. The reasons for this involve a deep involvement of both the experts and action researcher in the working practice of the company undergoing analysis and the evaluation of the proposed methods by applying them to this specific context.

There were two particular threats to the external validity in our studies. The first was that the proposed methods and tools would apply only to the automotive software development process at Volvo Cars and not to other automotive companies. Therefore, we included several other companies (both OEMs and suppliers) in the evaluation of the proposed methods and tools.

The second threat was related to the AUTOSAR meta-model that was the unit of analysis in case studies 2, 3 and 4. The proposed methods and tools that we applied to the AUTOSAR meta-model should also be applicable to other domain-specific meta-models. Therefore, we mapped the layers of the AUTOSAR modelling environment to the layers of MOF that are commonly accepted layers for modelling in different domains. In addition, we discussed the steps that must be taken in order to apply the proposed methods to meta-models of other domains, such as avionics, telecommunications and banking, in the thesis papers.

Construct validity

Construct validity concerns the mismatch between the theory and observations. In our studies, this was related to the ability of the measures to capture the desired system properties. We ensured this by the theoretical validation of measures presented in Case Study 1 and because the measures used in Case Study 3 were based on commonly-used UML measures previously proven in the scientific world. Finally, the measure of change used in case studies 2 and 4 was defined according to the GQM approach based on the conceptual model, which ensured that all relevant meta-model changes were captured.

Conclusion validity

Conclusion validity concerns the degree to which the conclusions of the studies are reasonable. In Case Study 3, this was related to the significance of the results obtained by the statistical analysis, which was high. In our other case studies, the conclusions were derived based on applying the methods to industrial scenarios and comparing the results with the expectations of experts. The conclusion was that the results could capture the desired properties.

1.6 Future work

Our future work will follow two major directions. Firstly, we plan to apply the methods and tools proposed in this thesis to the analysis of domain-specific meta-model evolution on meta-models from other, non-automotive industries. If these domain-specific meta-models are not available to us, we will apply the methods to the evolution of the UML meta-model.

Secondly, we plan to extend our analysis of the software model and meta-model evolution to other artifacts in the automotive development process, such as system requirements. In particular, we plan to analyze the co-evolution of the software models, meta-models and requirements in the automotive domain and answer research questions, such as: "*How can the architectural models and system requirements be evolved efficiently when adopting new meta-model versions in automotive development projects?*".

1.7 Personal contribution

The author of this thesis has been the main contributor as regards the planning and execution of the studies described in the thesis, and writing of the included publications.

Bibliography

[1] J. Bereisa. “Applications of Microcomputers in Automotive Electronics”. In: *IEEE Transactions on Industrial Electronics* 30.2 (1983).

[2] R. N. Charette. “This Car runs on Code”. In: *IEEE Spectrum Magazine* 46.3 (2009).

[3] P. Liggesmeyer and M. Trapp. “Trends in Embedded Software Engineering”. In: *Journal of IEEE Software* 26.3 (2009), pp. 19–25.

[4] *Automotive Open System Architecture.* AUTOSAR. www.autosar.org, 2003.

[5] C. Atkinson and T. Kühne. “Model-Driven Development: A Metamodeling Foundation”. In: *Journal of IEEE Software* 20.5 (2003), pp. 36–41.

[6] Jean Bézivin and Olivier Gerbé. “Towards a Precise Definition of the OMG/MDA Framework”. In: *Proceedings of the International Conference on Automated Software Engineering.* 2001, pp. 273–280.

[7] *Information Technology - Information Resource Dictionary System.* ISO/IEC 10027. 1990.

[8] C. Atkinson and T. Kühne. “Strict Profiles: Why and How”. In: *In Proceedings of the 3rd International Conference on the Unified Modeling Language, Lecture Notes in Computer Science.* 2000, pp. 309–322.

[9] C. Atkinson, T. Kühne, and B Henderson-Sellers. “To Meta or not to Meta - That is the Question”. In: *Journal of Object - Oriented Programming* 13.8 (2000), pp. 32–36.

[10] *Object Management Group.* OMG. www.omg.org, 1989.

[11] *OMG MOF 2.0 Core Final Adopted Specification.* Object Management Group. www.omg.org, 2004.

[12] A. Kleppe. “A Language Description is More than a Metamodel”. In: *Proceedings of the 4th International Workshop on Software Language Engineering.* 2007, pp. 273–280.

[13] C. Atkinson and T. Kühne. “Rearchitecting the UML Infrastructure”. In: *Journal of Transactions on Modeling and Computer Simulation* 12.4 (2002), pp. 291–321.

[14] C. Atkinson, T. Kühne, and B. Henderson-Sellers. “Stereotypical Encounters of the Third Kind”. In: *In Proceedings of the 5th International Conference on The Unified Modeling Language.* 2002, pp. 100–114.

[15] M. Staron, L. Kuzniarz, and C. Wohlin. “Empirical Assessment of Using Stereotypes to Improve Comprehension of UML Models: A Set of Experiments”. In: *Journal of Systems and Software* 79.5 (2006), pp. 727–742.

[16] T. Kühne. “Matters of (Meta-) Modeling”. In: *Journal of Software and Systems Modeling* 5.4 (2006), pp. 369–385.

[17] M. Saeki and H. Kaiya. “On Relationships among Models, Meta Models and Ontologies”. In: *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling.* 2007.

[18] G. Nordstrom et al. “Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments”. In: *Proceedings of the IEEE Conference on Engineering of Computer Based Systems.* 1999, pp. 68–74.

[19] R. Flatscher. “Metamodeling in EIA/CDIF - Meta-Metamodel and Metamodels”. In: *ACM Transactions on Modeling and Computer Simulation.* 2002, pp. 322–342.

[20] *OMG Object Constraint Language Version 2.0.* Object Management Group. www.omg.org, 2006.

[21] D Di Ruscio, L. Iovino, and A. Pierantonio. “Evolutionary Togetherness: How to Manage Coupled Evolution in Metamodeling Ecosystems”. In: *Proceedings of the 6th International Conference on Graph Transformations.* 2012, pp. 20–37.

[22] D. Mendez et al. “Towards Transformation Migration After Metamodel Evolution”. In: *Proceedings of the Model and Evolution Workshop.* 2010, pp. 20–37.

[23] F. Mantz, G. Taentzer, and Y. Lamo. “Well-formed Model Co-evolution with Customizable Model Migration”. In: *Proceedings of the International Workshop on Principles of Software Evolution.* 2013, pp. 1–10.

[24] U. Aßmann, S. Zschaler, and G. Wagner. “Ontologies, Meta-models, and the Model-Driven Paradigm”. In: *Ontologies for Software Engineering and Software Technology.* Springer Berlin Heidelberg, 2006, pp. 249–273.

[25] R. Taylor, N. Medvidovic, and E. Dashofy. *Software Architecture: Foundations, Theory, and Practice.* Wiley, 2009.

[26] N. Mellegård and M. Staron. “Characterizing Model Usage in Embedded Software Engineering: A Case Study”. In: *Proceedings of the European Conference on Software Architecture: Companion Volume.* 2010, pp. 245–252.

[27] *AUTOSAR Generic Structure Template v4.2.1.* www.autosar.org, 2013.

[28] *OMG XML Metadata Interchange (XMI) Version 2.2.* Object Management Group. www.omg.org, 2011.

[29] *Systems and Software Engineering - Product Quality.* ISO/IEC 9126-1. 1991.

[30] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach, 2nd edition.* London, International Thomson Computer Press, 1996.

[31] *Standard for a Software Quality Metrics Methodology*. IEEE. 1998.

[32] F. García et al. "Towards a Consistent Terminology for Software Measurement". In: *Journal of Information and Software Technology* 48.8 (2006).

[33] V. Basili, G. Caldiera, and H. Rombach. *The Goal Question Metric Approach*. Encyclopedia of Software Engineering, Wiley, 1994.

[34] *Systems and Software Engineering - Measurement Process*. ISO/IEC 15939. 2007.

[35] J. Rosenberg. "Some Misconceptions about Lines of Code". In: *Proceedings of the 4th International IEEE Symposium on Software Metrics*. 1997, pp. 137–142.

[36] M. Jørgensen. "Software Quality Measurement". In: *Journal of Advances in Engineering Software* 30.12 (1999), pp. 907–912.

[37] L. Briand, K. El Emam, and S. Morasca. *Theoretical and Empirical Validation of Software Product Measures*. Tech. rep. International software Engineering Research Network, 1995.

[38] B. Kitchenham, S. L. Pfleeger, and N. Fenton. "Towards a Framework for Software Measurement Validation". In: *Journal of IEEE Transactions on Software Engineering* 21.12 (1995), pp. 929–944.

[39] L.C. Briand, S. Morasca, and V.R. Basili. "Property-based Software Engineering Measurement". In: *IEEE Transactions on Software Engineering* 22.1 (1996), pp. 68–86.

[40] L. Mathiassen. "Collaborative Practice Research". In: *Journal of Information Technology & People* 15.4 (2002), pp. 321–345.

[41] K. Lewin. "Action Research and Minority Problems". In: *Journal of Social Issues* 2.4 (1946), pp. 34–46.

[42] R. Rapoport. "Three Dilemmas in Action Research". In: *Journal of Human Relations* 23.6 (1970), pp. 499–513.

[43] L. Dickens and K. Watkins. "Action Research: Rethinking Lewin". In: *Journal of Management Learning* 30.2 (1999), pp. 127–140.

[44] G. Susman and R. Evered. "An Assessment of the Scientific Merits of Action Research". In: *Journal of Administrative Science Quarterly* 23.4 (1978), pp. 582–603.

[45] I. Chein, S. Cook, and J. Harding. "The Field of Action Research". In: *Journal of American Psychologist* 3.2 (1948), pp. 43–50.

[46] R. Glass. "The Software Research Crisis". In: *Journal of IEEE Software* 11.6 (1994), pp. 42–47.

[47] R. Yin. *Case Study Research: Design and Methods, 5th edition*. London, SAGE, 2014.

[48] C. Robson. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers, 2nd edition*. Blackwell Oxford, 2002.

[49] L. Barriball and A. While. "Collecting Data Using a Semi-Structured Interview: A Discussion Paper". In: *Journal of Advanced Nursing* 19.2 (1994), pp. 328–335.

[50] M. Aliaga and B. Gunderson. *Interactive Statistics, end edition.* Prentice Hall, 1999.

[51] R. Baskerville and A T. Wood-Harper. “A Critical Perspective on Action Research as a Method for Information Systems Research”. In: *Journal of Information Technology* 11.2 (1996), pp. 235–246.

[52] T. Cook and D. Campbell. *Quasi-Experimentation: Design & Analysis Issues for Field Settings.* Houghton Mifflin, 1979.